# Project 1: Gold Chase (First iteration)

This project uses of shared memory and semaphores to coordinate multiple process. The game of Gold Chase involves 1-5 players moving around a map searching for gold. The map shows many potential locations for the gold, but only one location contains gold. The other locations contain fool's gold. To win the game, find the real gold and make it to the edge of the map.

## Rules

1. Use `h`,`j`,`k`,and `l` to move the player left, down, up, and right respectively.
2. If a player lands on top of a square containing gold, post a message declaring whether the player has found fool's gold or real gold.
3. A player may not move into a wall, or off the edge of the map. Silently ignore direction commands which attempt an illegal move.
4. One player may move over/through another player.
5. When a player who has found the real gold attempts to move off the map, he is declared winner--display a "You Won!" message. Note that in this first version, the other players won't know that a winner has been declared.
6. At any time a player may quit and leave the game by entering an upper-case `Q`.

## Technical requirements and notes

7. The map "drawn" beforehand in a text file using spaces and asterisks. The map is assumed to be a proper rectangle (no jagged edges). Ensure that you don't have trailing spaces!
8. The first line of the map file is a number, n, representing n-1 fool's gold + 1 gold which must be randomly placed on the map when the map is created in memory.
9. Internally, your map should be represented as an array of bytes (characters) in a shared memory segment.
10. The first game process started should create the shared memory segment and load the map. Subsequent processes should just utilize the existing map in shared memory. Additionally, the last process to exit should unlink the map.
11. The first process to start the game will become player 1. Subsequent processes will take the lowest available player number (the first process could quit, freeing up player 1 for the next starting process to claim). Currently active players will be determined through information stored in shared memory.
12. Each player should be randomly dropped onto the map when starting the game.
13. Each byte of map memory is maintained as 8 bits which can be turned on or off (i.e., don't store the map as ASCII values). Definitions for the bits (wall, gold, player, etc.) can be found in the header file `goldchase.h`. Use unsigned variables when working with bits.
14. To assist you with the rendering of the game map, a library, `libmap.a`, has been provided. Add `-lmap` (and `-L.` if necessary) to link to the map code.
15. The C++ source for the rendering code and a `Makefile` have been provided so that you may build the library on your own machine. For the purposes of this project, do not change the provided code--neither the source files nor the headers.

16. The underlying graphics technology used by the rendering code is ncurses and panels (an ncurses addon). These have libraries of their own. Therefore, in order for your code to link successfully, your must also link to these libraries: `-lncurses -lpanel`
17. Using shared memory requires the rt library: `-lrt`, and semaphores requires the pthread library: `-lpthread`
18. To avoid overwriting each other, only one process should be allowed to write to the shared memory at one time. Use a semaphore among the processes to ensure that only one process writes at a time. As with the shared memory, the first process should create the semaphore.
19. The Map class provides the interface you will use. The test program shows usage for each of the interface functions in the Map class.